МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

Курсовая РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: АВЛ-деревья-вставка и исключение. Исследование

Студент гр. 8304 _____ Масалыкин Д. Р.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург

2019

СОДЕРЖАНИЕ

Задание на курсовую работу

Студент Масалыкин Д. Р.

Группа 8304

Тема работы : АВЛ-деревья-вставка и исключение. Исследование

Исходные данные: Элементы дерева

Содержание пояснительной записки:

В отчёте должны быть:

- формальная постановка задачи;
- описание алгоритма;
- описание структур данных и функций;
- исследование - для вариантов с исследованием;
- выводы.
- программный код (в приложении);

В вариантах с исследованием вывод промежуточных данных не является строго обязательным, но должна быть возможность убедиться в корректности алгоритмов.

Дата выдачи задания: 11.10.2019

Дата сдачи реферата: 27.12.2019

Дата защиты реферата: 27.12.2019

Студент		Масалыкин Д.Р.
Преподаватель		Фирсов М.А.

АННОТАЦИЯ

В данной курсовой работе проводится исследование зависимостей высоты АВЛ-дерева, количества операций, выполняемых при вставки нового элемента, и количество операций, выполняемых при удалении определенных элементов из дерева, от количества элементов в самом дереве. В работе представлены как теоретические данные, так и практические, также приведены графики для их сравнения.

ВВЕДЕНИЕ

АВЛ-дерево — это прежде всего двоичное дерево поиска, ключи которого удовлетворяют стандартному свойству: ключ любого узла дерева не меньше любого ключа в левом поддереве данного узла и не больше любого ключа в правом поддереве этого узла. Это значит, что для поиска нужного ключа в АВЛ-дереве можно использовать двоичный алгоритм поиска. Все ключи в дереве целочисленны и не повторяются.

Особенностью АВЛ-дерева является то, что оно является сбалансированным в следующем смысле: для любого узла дерева высота его правого поддерева отличается от высоты левого поддерева не более чем на единицу.

1. Задание

АВЛ-деревья - вставка и исключение. Исследование (в среднем, в худшем случае)

2. Формальная постановка задачи

Основная цель которого в том, чтобы выяснить, каким образом высота дерева, количество операций, проводимых при удалении и вставки, зависят от количества элементов в дереве. Для этого генерируется определенное количество элементов, достаточное для исследования.

3. Описание алгоритма

1) Открывается файл *Tree.txt*, в котором содержатся элементы бинарного дерева.
2) Считывается информация из файла и с помощью функции *insert* каждый элемент последовательно добавляется в бинарное дерево поиска таким образом, чтобы не нарушалось правило построения АВЛ-дерева(ключ любого узла дерева не меньше любого ключа в левом поддереве данного узла и не больше любого ключа в правом поддереве этого узла).
3) После вставки элемента дерево проверяется на сбалансированность (для любого узла дерева высота его правого поддерева отличается от высоты левого поддерева не более чем на единицу). Если дерево не

сбалансированно, то вызывается функция *balance,* которая работает следующим образом:

a) Если разница высот правого и левого поддеревьев равна 2, выполняется правый поворот.

b) Если разница высот правого и левого поддеревьев равна -2, выполняется левый поворот.

c) Если высота правого поддерева на 2 больше высоты левого поддерева, выполняется либо простой поворот влево вокруг p, либо так называемый *большой поворот* влево вокруг того же узла.

4) После того как бинарное дерево поиска считано из файла и построено АВЛ-дерево, пользователю предлагается выбрать элемент и удалить его из дерева. Для этого вызывается функция *remove*, которая находит узел с заданным ключом (бинарный поиск) и min узел с наименьшим ключом в правом поддереве, а затем заменяет удаляемый узел p на найденный узел min.

5) Если найденный узел не имеет правого поддерева, то по свойству АВЛ-дерева слева у этого узла может быть только один единственный дочерний узел (дерево высоты 1), либо узел вообще является листом. В обоих этих случаях функция просто удаляет узел и возвращает в качестве результата указатель на левый дочерний узел найденного узла.

4. Описание структур данных и функций

4.1 Описание функций:

1) unsigned char height(node p)*

Входные данные: указатель на корневой узел дерева.

Функция работы с высотой.

Возвращаемое значение: высота текущего узла

2) int bfactor(node p)*

Входные данные: указатель на узел.

Функция вычисляет balance factor заданного узла (и работает только с ненулевыми указателями)

Возвращаемое значение: разность между высотой правого и левого поддерева

3) void fixheight(node p)*

Входные данные: указатель на узел

Функция восстанавливает корректное значение поля height заданного узла (при условии, что значения этого поля в правом и левом дочерних узлах являются корректными)

Возвращаемое значение: нет.

4) *node* rotateright(node* p)*

Входные данные: указатель на узел

Функция реализует правый поворот дерева.

Возвращаемое значение: указатель на новый корневой узел.

5) *node* rotateleft(node* q)*

Входные данные: указатель на узел.

Функция реализует левый поворот дерева.

Возвращаемое значение: указатель на новый корневой узел.

6) *node* balance(node* p)*

Входные данные: указатель на узел.

Функция проверяет разность высот между левым и правым поддеревом и выполняет нужный поворот.

Возвращаемое значение: указатель на корневой узел.

7) *node* insert(node* p, int k)*

Входные данные: *p*-указатель на корневой узел, *k*-ключ вставляемого элемента.

Функция находит место вставки ключа *k*, вставляет его, а затем вызывает функцию балансировки.

Возвращаемое значение: указатель на корневой узел.

8) *node* findmin(node* p)*

Входные данные: указатель на узел.

Функция выполняет поиск узла с минимальным ключом в дереве p.

Возвращаемое значение: указатель на узел с минимальным ключом.

9) *node* removemin(node* p)*

Входные данные: указатель на узел.

Функция удаляет минимальный элемент из заданного дерева.

Возвращаемое значение: указатель на корневой узел.

10) *node* remove(node* p, int k)*

Входные данные: *p*-указатель на узел, *k*- ключ удаляемого элемента.

Функция находит элемент в дереве, удаляет его и вызывает функцию балансировки.

Возвращаемое значение: указатель на корневой узел.

11)node generateTree(node* p, int N)*

Входные данные: *p*-указатель на корневой узел, *N*-номер генерируемого дерева.

Функция генерирует деревья, в которых содержатся элементы от 0 до 10000.

Возвращаемое значение: указатель на корневой узел.

4.2 Описание структур данных:

1) struct node

   Структура содержит:

   - ✓ int key - ключ узла;
   - ✓ unsigned char height-высоту поддерева с корнем в данном узле
   - ✓ struct node* left - указатель на левое поддерево;
   - ✓ struct node* right - указатель на правое поддерево;

6. Исследование

6.1 Зависимость высоты дерева от количества элементов

Теорема. (Г. М. Адельсон-Вельский и Е. М. Ландис). Высота сбалансированного дерева с N внутренними узлами заключена между $\log_2(N+1)$ и $1.4404\log_2(N+2)-0.328$.

Для практического исследования зависимости высоты дерева от количества элементов генерировался массив из случайно расположенных чисел от 1 до 10000, далее эти числа последовательно вставлялись в изначально пустое АВЛ-дерево и измерялась высота дерева после каждой вставки. Данные результаты были усреднены по 1000 расчетам и получены средние значения высот. Также для каждого n (0<n<10000) были найдены минимальные и максимальные высоты.

На следующем графике показана зависимость от n средней высоты (желтая линия); минимальной высоты (голубая линия); максимальной высоты (красная линия). Кроме того, показаны верхняя и нижняя теоретические оценки(зеленая и фиолетовая линии соответственно).

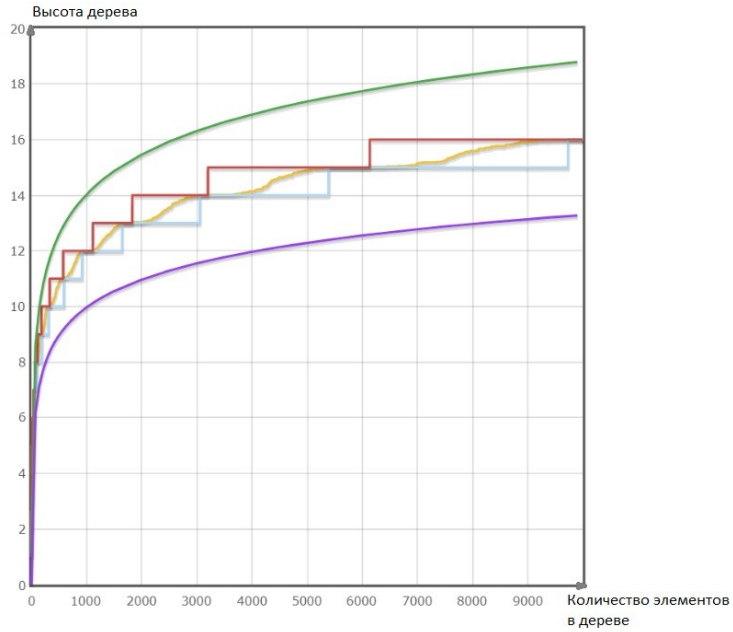


Рис. 1 Зависимость высоты АВЛ-дерева от количества элементов в дереве

6.2 Зависимость числа операций при вставке нового элемента от количества элементов в дереве

Вставка нового ключа в АВЛ-дерево выполняется так же, как это делается в простых деревьях поиска: спускаемся вниз по дереву, выбирая правое или левое направление движения в зависимости от результата сравнения ключа в текущем узле и вставляемого ключа. Таким образом, требуется $\log_2 N$ операций для нахождения места вставки элемента и еще одна операция для вставки элемента. После вставки нового узла каждый предшествующий вставленному элементу узел проверяется на сбалансированность. Будем считать, что совершается очередная операция, если проверенный узел разбалансирован и требует балансировки.

Согласно предшествующему описанию, теоретически минимально возможное количество операций для вставки элемента в дерево $\log_2 N+1$ ($\log_2 N$ для нахождения места вставки и 1 операция для самой вставки).

Максимально же возможное количество операций потребуется в том случае, когда все предстоящие узлы окажутся разбалансированы. В таком случае понадобится ещё $\log_2 N+1$ операций для того, чтобы сбалансировать каждый узел дерева. В сумме получили $2\log_2 N+2$ максимально возможных операций.

Для практического исследования зависимости числа операций при вставке нового элемента от количества элементов в дереве генерировался массив из случайно расположенных чисел от 1 до 10000, далее эти числа последовательно вставлялись в изначально пустое АВЛ-дерево и измерялось количество операций, произведенных для вставки элемента. Данные результаты были усреднены по 1000 расчетам и получены средние значения высот. Также для каждого n (0<n<10000) были найдены минимальное и максимальное количество операций .

На следующем графике показана зависимость среднего количества операций при вставке нового элемента от количества элементов в дереве(зеленая линия); минимальное (фиолетовая линия) и максимальное (голубая линия) количество операций. Кроме того, показаны верхняя и нижняя теоретические оценки (красная и желтая линии соответственно).

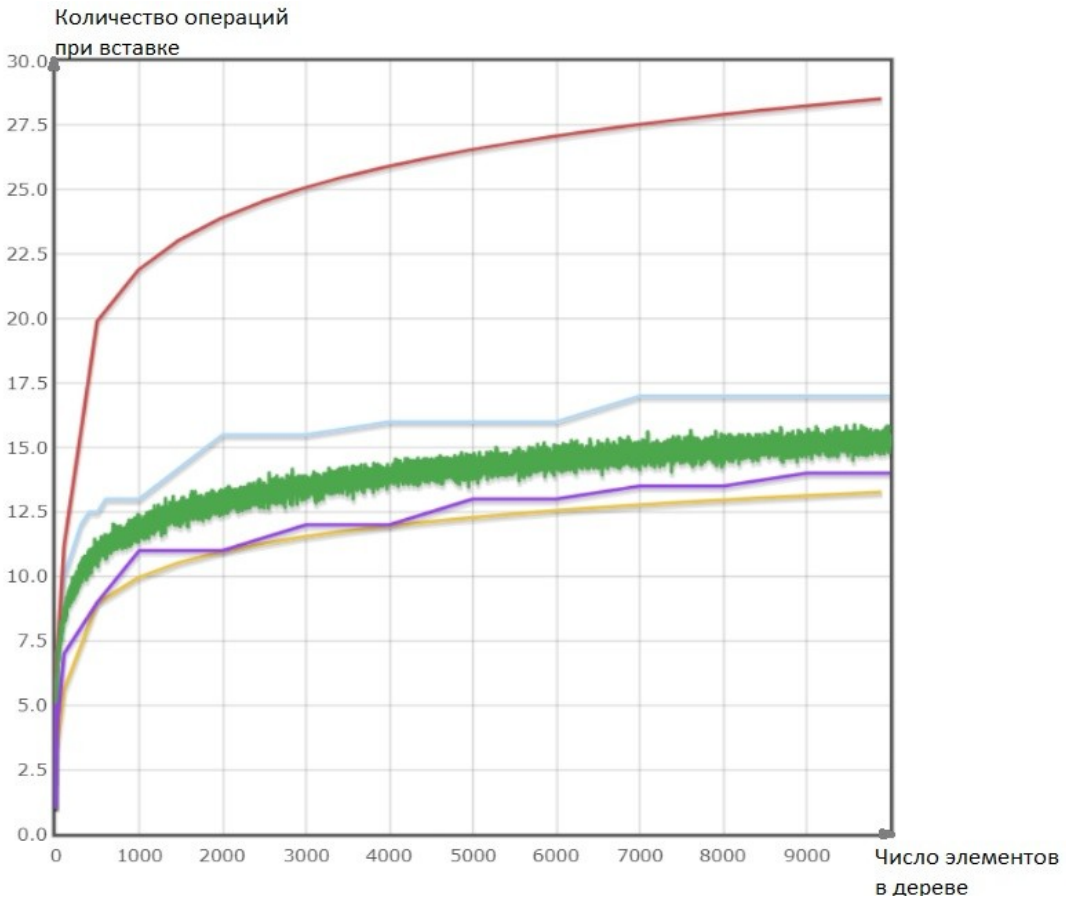


Рис. 2 Зависимость количества операций при вставки элемента в АВЛ-дерево от количества элементов дереве

6.3 Зависимость числа операций при удалении элемента от количества элементов в дереве

Для удаления заданного элемента с ключом k требуется найти некоторый узел p, соответствующий данному ключу. После нахождения самого элемента требуется в его правом поддереве найти узел min с наименьшим ключом и заменить удаляемый узел p на найденный узел min. Поскольку АВЛ-дерево в плане поиска соответствует двоичному дереву поиска, то операция нахождения самого элемента, а затем минимального в его правом поддереве выполняется за $\log_2 N$ операций.

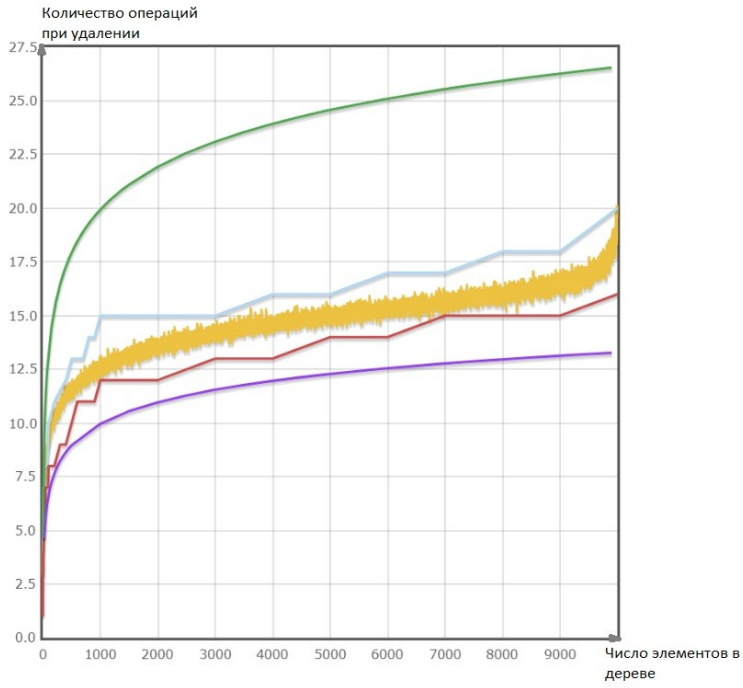
Как только элемент был найден, он удаляется. Будем считать операцию удаления как одну операцию. После удаления узла каждый предшествующий удаленному элементу узел проверяется на сбалансированность. Считаем, что совершается очередная операция, если проверенный узел разбалансирован и требует балансировки.

При реализации возникает несколько нюансов. Прежде всего, если найденный узел p не имеет правого поддерева, то по свойству АВЛ-дерева слева у этого узла может быть только один единственный дочерний узел (дерево высоты 1), либо узел p вообще лист. В обоих этих случаях надо узел p просто удаляется и возвращается в качестве результата указатель на левый дочерний узел узла p.

Согласно предшествующему описанию, теоретически минимально возможное количество операций для удаления элемента из дерева $\log_2 N+1$ ($\log_2 N$ для нахождения элемента с ключом k, а затем элемента с минимальным ключом в левом поддереве и 1 операция для удаления). Максимально же возможное количество операций потребуется в том случае, когда после удаления элемента все предстоящие узлы окажутся разбалансированы. В таком случае понадобится ещё $\log_2 N$ операций для того, чтобы сбалансировать каждый узел дерева. В сумме получили $2\log_2 N+1$ максимально возможных операций.

Для практического исследования зависимости числа операций при удалении элемента от количества элементов в дереве генерировался массив из случайно расположенных чисел от 1 до 10000, далее эти числа последовательно вставлялись в изначально пустое АВЛ-дерево и генерировались случайные ключи. Согласно данным ключам удалялись элементы из дерева и измерялось количество операций, произведенных для удаления элемента и дальнейшей балансировки дерева. Данные результаты были усреднены по 1000 расчетам и получены средние значения высот. Также для каждого n (0<n<10000) были найдены минимальное и максимальное количество операций .

На следующем графике показана зависимость среднего количества операций при удалении элемента от количества элементов в дереве(желтая линия); минимальное (красная



линия) и максимальное (голубая линия) количество операций. Кроме того, показаны верхняя и нижняя теоретические оценки (зеленая и фиолетовые линии соответственно).

Рис. 3 Зависимость количества операций при удалении элемента из АВЛ-дерева от количества элементов в дереве

Заключение

В ходе исследования было выяснено, что операции вставки и удаления (а также более простая операция поиска) выполняются за время пропорциональное высоте дерева, т.к. в процессе выполнения этих операций производится спуск из корня к заданному узлу, и на каждом уровне выполняется некоторое фиксированное число действий. В ходе исследования были получены результаты, лишь немного превышающие минимальные теоретические показатели. Из этого можно сделать вывод об эффективности АВЛ-дерева т. к. основные операции(удаление, вставка и поиск) выполняются за время немногим больше минимально возможного. А в силу того, что АВЛ-дерево является сбалансированным, его высота зависит логарифмически от числа узлов и практически не отличается от теоретических показателей. Такая зависимость позволяет ускорить выполнение основных операций, ведь нет необходимости проходить все элементы дерева. Таким образом, время выполнения всех базовых операций гарантированно логарифмически зависит от числа узлов дерева. Полученные результаты соотносятся с теорией.

Список использованных источников

1. B. Pfaff, An Introduction to Binary Search Trees and Balanced Trees.
2. Н. Вирт, Алгоритмы и структуры данных .
3. Т. Кормен и др., Алгоритмы: построение и анализ.
4. Д. Кнут, Искусство программирования. Том 3 Сортировка и поиск.
5. Статья АВЛ-деревья (сайт www.habrahabr.ru).

Приложение А. Код программы.

Main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])

{

 QApplicatio
 n a(argc,
 argv);
 MainWindo
 w w;

w.show();

return a.exec();

}
```

mainwindow.cpp

```
 #include
 "mainwind
 ow.h"
 #include
 "ui_mainwi
 ndow.h"

MainWindow::MainWindow(QWidget *parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

 ui-
 >line_inse
 rt-
 >clear();
 ui-
 >line_rem
 ove-
 >clear();
```

```
 scene = new
 QGraphicsScene(this); ui-
 >visualiztion->setScene(scene);
 file.open("research.txt");
}
 MainWindow::~MainWindow()
{
 delete ui;
 delete
 scene;
 file.close()
 ;
}
void MainWindow::on_button_insert_clicked()
{
 begin =
 clock();
 scene-
 >clear();
 head = insert(head, int(stoi(ui->line_insert->text().toStdString())));
 print_tree(head, ui->visualiztion->geometry().x() / 2, 25); str_tree.clear();
 str_tree = "(";
 txt_tree(head, str_tree);
 ui->avl_tree->setText(QString::fromStdString(str_tree));
 end = clock();
file<<"Insert"<<endl<<end-begin<<endl;
}
void MainWindow::on_button_remove_clicked()
{
 scene-
 >clear();
 begin =
 clock();
 head = remove(head, int(stoi(ui->line_remove->text().toStdString()))); end
 = clock();
 print_tree(head, ui->visualiztion->geometry().x() / 2, 25);
 file<<"Remove"<<endl<<end-begin<<endl;
```

```
 str_tree.cl
 ear();
 str_tree =
 "(";
 txt_tree(h
 ead,
 str_tree);
ui->avl_tree->setText(QString::fromStdString(str_tree));
}
void MainWindow::on_clear_clicked()
{
 foreach (QGraphicsItem* item,
 scene->items()) { delete item;
}
 head =
 delete_tre
 e(head);
 str_tree.cl
 ear();str_t
 ree = "(";
 ui-
 >line_inse
 rt-
 >clear();
 ui-
 >line_rem
 ove-
 >clear();
 ui-
 >avl_tree-
 >clear();
}
void MainWindow::on_actionsave_triggered()
{
 QString path = QFileDialog::getOpenFileName(this, tr("Open
 path to save"), "/home/egor/Desktop");
```

```
 if (path == nullptr) return;
 ofstream file(path.toStdString());
file << (str_tree == "(" ? "( )" : str_tree);
}
void MainWindow::on_actionopen_triggered()
{
 QString path = QFileDialog::getOpenFileName(this, tr("Open path to download tree"), "/
 home/egor/Desktop");
 if (path == nullptr) return; ifstream
 oFile(path.toStdString()); string str;
oFile >> str;
for (unsigned long i = 0; i < str.size(); i++)
{
if (str[i] == '(' || str[i] == ')' || str[i] == '#')
{
str[i] = ' ';
}
}
 stringstream
 sstream; int tmp;
sstream << str;
 head =
 delete_tree(head);
 str_tree.clear();
ui->avl_tree->clear();
 ui->line_insert-
 >clear(); ui-
 >line_remove->clear();
 str_tree = "(";
 while(sstream >> tmp)
{
head = insert(head, tmp);
}
 txt_tree(head, str_tree);
```

```
 ui->avl_tree-
 >setText(QString::fromStdString(str_tree));
 print_tree(head, ui->visualiztion->geometry().x()
 / 2, 25);
}
void MainWindow::on_actioninfo_triggered()
{
(new HelpBrowser (":/docs/doc", "index.htm"))->show();
}
void MainWindow::print_tree(node* tree, int x, int y)
{
 if (!
 tree)
 return;
 const
 int
 offset
 = 30;
 const
 int r =
 25;
if (tree->left)
{
 QLine line(x + r, y + r, x - offset * tree->left->height * tree->left-
 >height + 10, y + 90);
scene->addLine(line, QPen(Qt::black));
}
if (tree->right)
{
 QLine line(x + r, y + r, x + offset * tree->right->height * tree->right-
 >height + 35, y + 90);
scene->addLine(line, QPen(Qt::black));
```

```
}
 scene->addEllipse(x, y, 2 * r, 2 * r , QPen(Qt::black), QBrush(Qt::white)); int temp =
 tree->key;
 int count_zero = 1;
 while (temp /= 10)
 count_zero++;
 QString zeroes = (tree->key >= 0 ? "" : "-"); for
 (int i = 0; i < 4 - count_zero; i++) zeroes += '0';
QGraphicsTextItem* txtItem =
 new QGraphicsTextItem(zeroes + QString::number(abs(tree->key))); if
 (tree->key >= 0)
 txtItem->setPos(x + 7, y + 15); else
 txtItem->setPos(x + 4, y + 15);
 scene->addItem(txtItem);
if (tree->left) {
 print_tree(tree->left, x - offset * tree->left->height * tree->left->height, y + 75);
}
if (tree->right) {
print_tree(tree->right, x + offset * tree->right->height * tree->right->height, y
+ 75);
}
}
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define
MAINWINDOW_H
#include <QMainWindow>
#include <QFileDialog>
#include
<QGraphicsScene>
#include <QGraphicsTextItem>
#include <string>
#include <fstream>
#include <ctime>
#include <fstream>
#include "avl_tree.h"
#include "help.h" using
namespace std;
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
```

```
class MainWindow : public QMainWindow
 { Q_OBJ
 ECT
public:
MainWindow(QWidget *parent = nullptr);
 ~MainWindow(
 ); private slots:
 void on_button_insert_clicked();
 void on_button_remove_clicked();
 void on_actionsave_triggered();
 void on_clear_clicked();
 void on_actionopen_triggered();
 void on_actioninfo_triggered();
 //void on_actionabout_triggered();
 private:
Ui::MainWindow *ui;
 //About *abt;
 QGraphicsScene* scene;
 void print_tree(node*, int, int); node*
 head = nullptr;
 QString qstr_tree = "";
 string str_tree = "(";
 time_t begin;
```

```
 time_t end;
 ofstream file;
};
#endif // MAINWINDOW_H
```

avl_tree.cpp

```
#include "avl_tree.h" unsigned
char height(node* p)
{
return p ? p->height : 0;
}
int bfactor(node* p)
{
return height(p->right) - height(p->left);
}
void fixheight(node* p)
{
 unsigned char hl = height(p->left);
 unsigned char hr = height(p->right); p-
>height = (hl > hr ? hl : hr) + 1;
}
node* rotateright(node* p) // правый поворот вокруг p
{
 node* q = p->left; p-
>left = q->right; q-
>right = p;
 fixheight(p);
 fixheight(q); return
 q;
}
node* rotateleft(node* q) // левый поворот вокруг q
{
 node* p = q->right;
 q->right = p->left; p-
>left = q;
 fixheight(q);
 fixheight(p); return
 p;
}
node* balance(node* p) // балансировка узла p
{
```

```
fixheight(p);
if( bfactor(p) == 2 )
{
if( bfactor(p->right) < 0 )
 p->right = rotateright(p->right);
 return rotateleft(p);
}
if( bfactor(p) == -2 )
{
if( bfactor(p->left) > 0 )
 p->left = rotateleft(p->left); return
 rotateright(p);
}
return p; // балансировка не нужна
}
node* insert(node* p, int k) // вставка ключа k в дерево с корнем p
{
 if( !p ) return new node(k); if( k
 < p->key )
 p->left = insert(p->left, k); else
 p->right = insert(p->right, k);
 return balance(p);
}
node* findmin(node* p) // поиск узла с минимальным ключом в дереве p
```

```
{
return p->left ? findmin(p->left) : p;
}
node* removemin(node* p) // удаление узла с минимальным ключом из дерева p
{
 if( p->left == nullptr )
 return p->right;
 p->left = removemin(p->left);
 return balance(p);
}
node* remove(node* p, int k) // удаление ключа k из дерева p
{
 if( !p ) return nullptr; if( k
 < p->key )
 p->left = remove(p->left,k);
 else if( k > p->key )
 p->right = remove(p->right,k);
 else // k == p->key
{
 node* q = p->left;
 node* r = p->right;
 delete p;
 if( !r ) return q; node*
 min = findmin(r);
 min->right = removemin(r);
 min->left = q;
return balance(min);
}
return balance(p);
}
node* delete_tree(node* tree)
{
 if (!tree) return nullptr;
 delete_tree(tree->left);
 delete_tree(tree->right);
 delete tree;
return nullptr;
}
```

```
void txt_tree(node* tree, string& str)
{
if (!tree)
{
 str = "";
 return;
}
 stringstream sstream;
 string tmp;
 sstream << tree->key;
 sstream >> tmp;
str += tmp;
if (tree->left)
{
str += "(";
txt_tree(tree->left, str);
} else if (tree->right)
{
str += "(#)";
}
if (tree->right)
{
txt_tree(tree->right, str);
}
str += ")";
}
```

avl_tree.h

```
#ifndef
AVL_TREE_H
#define
AVL_TREE_H
#include <string>
#include <sstream>
using namespace std;
struct node // структура для представления узлов дерева
{
int key;
unsigned char height;
node* left;
node* right;
explicit node(int k) { key = k; left = right = nullptr; height = 1; }
};
unsigned char height(node* p); // определение высоты дерева int
bfactor(node* p); // баланс фактор
void fixheight(node* p); // подсчет высоты
node* rotateright(node* p); // правый поворот вокруг p node*
rotateleft(node* q); // левый поворот вокруг q
node* insert(node* p, int k); // вставка ключа k в дерево с корнем p node*
findmin(node* p); // поиск узла с минимальным ключом в дереве p
node* removemin(node* p); // удаление узла с минимальным ключом из дерева p
node* remove(node* p, int k); // удаление ключа k из дерева p
node* delete_tree(node* tree);// удаление дерева
void txt_tree(node* tree, string& str); // скобочная запись дерева #endif //
AVL_TREE_H
```

help.h

```
#ifndef HELP_H
#define HELP_H
#include <QtWidgets>
class HelpBrowser : public QWidget {
Q_OBJECT
public:
HelpBrowser(const QString& strPath,
const QString& strFileName,
QWidget* pwgt = nullptr
) : QWidget(pwgt)
{
QPushButton* pcmdBack = new QPushButton("<<");
QPushButton* pcmdHome = new
```

```
 QPushButton("Home"); QPushButton* pcmdForward =
 new QPushButton(">>"); QTextBrowser* ptxtBrowser
 = new QTextBrowser; setMinimumSize(400, 400);
 connect(pcmdBack, SIGNAL(clicked()),
 ptxtBrowser, SLOT(backward())
);
 connect(pcmdHome,
 SIGNAL(clicked()), ptxtBrowser,
 SLOT(home())
);
 connect(pcmdForward, SIGNAL(clicked()),
 ptxtBrowser, SLOT(forward())
);
 connect(ptxtBrowser, SIGNAL(backwardAvailable(bool)),
 pcmdBack, SLOT(setEnabled(bool))
);
 connect(ptxtBrowser, SIGNAL(forwardAvailable(bool)),
 pcmdForward, SLOT(setEnabled(bool))
);
 ptxtBrowser->setSearchPaths(QStringList() << strPath);
 ptxtBrowser->setSource(QString(strFileName));
//Layout setup
 QVBoxLayout* pvbxLayout = new
 QVBoxLayout; QHBoxLayout* phbxLayout =
 new QHBoxLayout; phbxLayout-
 >addWidget(pcmdBack); phbxLayout-
 >addWidget(pcmdHome); phbxLayout-
 >addWidget(pcmdForward);
```

```
 pvbxLayout->addLayout(phbxLayout);
 pvbxLayout->addWidget(ptxtBrowser);
 setLayout(pvbxLayout);
}
};
#endif // HELP_H
```

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>MainWindow</class>
<widget class="QMainWindow" name="MainWindow">
<property name="geometry">
<rect>
<x>0</x>
<y>0</y>
<width>762</width>
<height>522</height>
</rect>
</property>
<property name="windowTitle">
<string>MainWindow</string>
</property>
<widget class="QWidget" name="centralwidget">
<layout class="QGridLayout" name="gridLayout">
<item row="1" column="1" rowspan="2">
<widget class="QSpinBox" name="line_insert">
<property name="buttonSymbols">
<enum>QAbstractSpinBox::NoButtons</enum>
</property>
<property name="minimum">
<number>-9999</number>
</property>
<property name="maximum">
<number>9999</number>
</property>
```

```
</widget>
</item>
<item row="1" column="2">
<widget class="QLineEdit" name="avl_tree">
<property name="readOnly">
<bool>true</bool>
</property>
</widget>
</item>
<item row="1" column="0" rowspan="2">
<widget class="QPushButton" name="button_insert">
<property name="inputMethodHints">
<set>Qt::ImhDigitsOnly</set>
</property>
<property name="text">
<string>insert</string>
</property>
</widget>
</item>
<item row="0" column="0" colspan="7">
<widget class="QGraphicsView" name="visualiztion"/>
</item>
<item row="2" column="2">
<widget class="QPushButton" name="clear">
<property name="text">
<string>clear</string>
</property>
</widget>
</item>
<item row="1" column="3" rowspan="2">
```

<widget class="QSpinBox" name="line_remove">
<property name="buttonSymbols">
<enum>QAbstractSpinBox::NoButtons</enum>
</property>
<property name="minimum">
<number>-9999</number>
</property>
<property name="maximum">
<number>9999</number>
</property>
</widget>
</item>
<item row="1" column="5" rowspan="2">
<widget class="QPushButton" name="button_remove">
<property name="text">
<string>remove</string>
</property>
</widget>
</item>
</layout>
</widget>
<widget class="QStatusBar" name="statusbar"/>
<widget class="QMenuBar" name="menubar">
<property name="geometry">
<rect>
<x>0</x>
<y>0</y>
<width>762</width>
<height>26</height>
</rect>
</property>
</widget>
<widget class="QToolBar" name="toolBar">
<property name="windowTitle">

```
<string>toolBar</string>
</property>
<attribute name="toolBarArea">
<enum>TopToolBarArea</enum>
</attribute>
<attribute name="toolBarBreak">
<bool>false</bool>
</attribute>
<addaction name="actionsave"/>
<addaction name="actionopen"/>
<addaction name="actioninfo"/>
</widget>
<action name="actionsave">
<property name="text">
<string>save</string>
</property>
</action>
<action name="actionopen">
<property name="text">
<string>open</string>
</property>
</action>
<action name="actioninfo">
<property name="text">
<string>info</string>
</property>
</action>
<action name="actionabout">
<property name="text">
<string>about</string>
</property>
```

```
</action>
</widget>
<resources/>
<connections/>
</ui>
```